УДК 004.422+004.428

ПРИМЕНЕНИЕ ОБОБЩЁННЫХ КОНЦЕПЦИЙ КАК БАЗОВЫХ АБСТРАКЦИЙ ДЛЯ КОМПОНЕНТОВ ДОСТУПА К УНИВЕРСАЛЬНОЙ МОДЕЛИ ХРАНЕНИЯ ДАННЫХ

Д. Е. Яблоков, ведущий инженер-программист Межвузовского научно-исследовательского центра по теоретическому материаловедению, аспирант Самарского национального исследовательского университета имени академика С. П. Королёва, г. Самара; dyablokov@gmail.com

Рассматриваются вопросы, связанные с созданием компонентов доступа к универсальному хранилищу данных. В качестве базовых абстракций предлагается применение обобщённых концепций, основанных на использовании нескольких парадигм программирования. Исследование проводится в рамках проекта создания экспертной системы с использованием информационной базы программного комплекса ToposPro. ***Целью статьи*** *является исследование возможной стратегии конструирования самостоятельных компонентов доступа и обработки данных, размещаемых в универсальном хранилище. Применение такой стратегии позволяет связать структуры хранения и средства доступа к данным с рассматриваемыми в статье обобщёнными концепциями адаптеров контейнерных классов и итераторов. При этом главными инструментами в процессе проектирования и реализации становятся обобщённые концепции, а конкретные классы на их основе могут настраиваться за счёт перегрузки, агрегирования, наследования и спецификации требований к абстрактным типам данных.*

Ключевые слова*: универсальная модель данных, ToposPro, парадигмы программирования, развитие концепций, абстракция данных, обобщённые концепции итераторов, адаптеры контейнерных классов.*

DOI: 10.21667/1995-4565-2016-57-3-68-74

Введение

Сфера компьютерных наук, которая возникла сравнительно недавно, получила стремительное развитие от состояния, когда в ней был занят лишь небольшой круг специалистов, до состояния, когда речь идёт о повсеместном использовании компьютерных технологий и связанных с ними результатов исследований в той или иной предметной области. Быстрый рост этого направления оказал влияние и на современное состояние индустрии программирования, которую сейчас нельзя представить без качественных технологий построения абстракций и базовых принципов, дающих основу соответствующему стилю проектирования, а также инструментальных средств, поддерживающихся используемыми парадигмами [1, 2], которые определяют стратегию конструирования программного обеспечения. И как следствие – обдуманное и правильное использование подобного рода механизмов и связанной с ними терминологии способствует формированию каркаса понятий, определяющего логические рамки для описания объектов или их семейств, работа с которыми становится возможной в терминах используемого понятийного аппарата. Это приводит к необходимости формирования набора абстракций, определяющих семантическое и синтаксическое поведение объектов, а также к появлению семейства обобщённых концепций [3], содержащих набор требований и дающих представление о свойствах и ограничениях, которыми должна обладать абстракция, являющаяся моделью одной или нескольких концепций.

Теоретическая часть

Моделирование – это процесс, определяющий отношение между концепцией и представляющей её семантические и синтаксические свойства абстракцией. Отношения же между концепциями формализуются процессом развития одной концепции от другой, когда развивающая концепция предоставляет какую-то часть функциональности развиваемой концепции и, возможно, некоторую дополнительную, являющуюся только частью развивающей (рисунок 1).

Абстракция может быть моделью более чем одной концепции, а последняя, в свою очередь, может быть развитием нескольких концепций.

Практически это означает, что если некоторый алгоритм требует, чтобы его формальные параметры являлись моделью какой-либо концепции, то его фактические параметры могут представлять собой тип, являющийся моделью другой концепции, при условии, что она развивает концепцию, определяющую набор требований к формальным параметрам. В этих терминах можно определить базовые принципы объектно-ориентированной и обобщённой парадигм программирования [4], такие как наследование, полиморфизм и спецификация требований к типу или семейству типов [5].

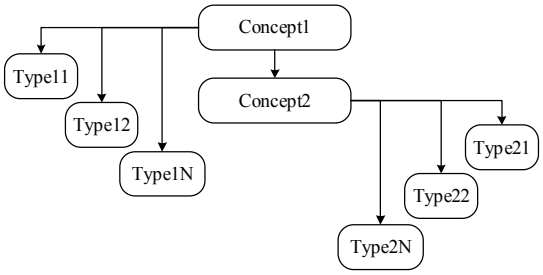

Рисунок 1 – Моделирование концепций

При разработке очень большой и очень сложной компьютерной программы, связанной с универсальным хранением данных, нужно потратить значительные усилия на уяснение и определение задачи, на осознание её сложности и разбиение её на меньшие подзадачи, решение которых легко реализовать. Немаловажным фактором при этом является необходимость использования соответствующих парадигм и проверенных временем идиом, основанных на опыте и твёрдом понимании выбранного языка программирования и объединении наилучших практических решений, рекомендаций и правил, определяющих стандартизированный способ получения на выходе наиболее качественного результата. Одним из примеров реализации такого стандартизированного набора рекомендаций и правил является обобщённая библиотека расширений (GLEX), разрабатываемая в рамках проекта создания экспертной системы на базе информационной среды программного комплекса ToposPro [6]. Её теоретическую основу составляет набор базовых концепций и требований к типам данных стандартной библиотеки шаблонов (STL) [7, 8], широко используемой при программировании на языке C++. В дальнейшем планируется широкое применение GLEX при реализации компонентов обработки и представления универсальных данных большого объёма. GLEX – это не просто библиотека, а скорее набор соглашений, объединяющих компоненты нескольких типов: адаптеры контейнеров, алгоритмы, концепции итераторов и функциональных адаптеров. Идея состоит в том, что адаптеры контейнеров и алгоритмы, работающие с ними, могут ничего не знать друг о друге. Это преимущество достигается за счёт обобщённых концепций итераторов.

Обобщённые концепции итераторов (рисунки 2 – 4) важны при проектировании библиотечных компонентов обработки и анализа данных, так как абстракции, построенные на базе этих понятий, могут являться интерфейсами между обобщёнными алгоритмами и адаптерами структур хранения данных. Они также имеют большое значение, поскольку являются обобщением указателей, т.е. объектов, которые указывают на другие объекты и могут использоваться как основа для создания компонентов, осуществляющих проход по диапазону. Если модель концепции итератора указывает на какую-либо позицию диапазона, то после применения операции продвижения (инкрементирование или декрементирование) она будет указывать позицию, являющуюся следующей или предыдущей, в зависимости от направления обхода. Нужно отметить, что взаимосвязь концепций итераторов со свойствами указателей не совсем однозначна. Например, указатели в языке C++ имеют очень развитую семантику и к ним применимы операции адресной арифметики [7], операция разыменования и т.п., но обобщённые алгоритмы на диапазонах в большинстве случаев используют лишь малое подмножество свойств указателей, другие же обобщённые алгоритмы используют иные подмножества. Таким образом, существует несколько различных способов обобщения семантики указателей, при этом каждый из способов является отдельной обобщённой концепцией.

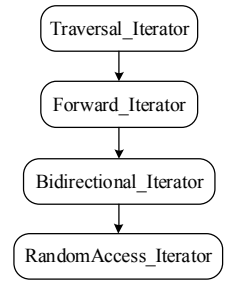

Рисунок 2 – Развитие концепций итераторов обхода

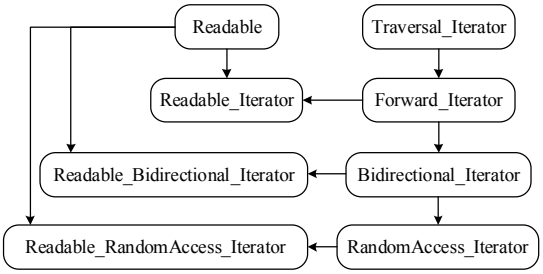

Рисунок 3 – Развитие концепций неизменяющих итераторов

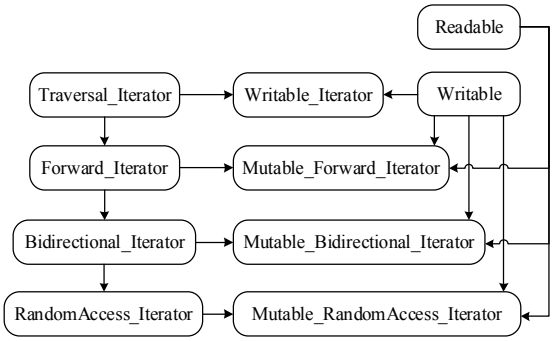

Рисунок 4 – Развитие концепций изменяющих итераторов

Понятие адаптирующей концепции итератора (рисунки 5 – 7) вводится для того, чтобы изменить поведение одной из уже имеющихся обобщённых концепций итераторов. Для этого необходимо создать адаптивный интерфейс, который, в определённом смысле, будет изменять поведение адаптирующей модели в терминах адаптируемой концепции. Важным аспектом применения адаптирующей модели является тот факт, что становится возможным сделать что-то вроде того, что было реализовано в качестве адаптируемой концепции, но с конкретным альтернативным поведением, например, изменяющим направление обхода диапазона или контекст отдельных операций. В общем случае это гарантирует, что обобщённые алгоритмы на диапазонах [7] будут правильно и эффективно работать с адаптирующими моделями итераторов, построенными на базе обобщённых концепций со строгими требованиями к семантике и формализованными интерфейсами.

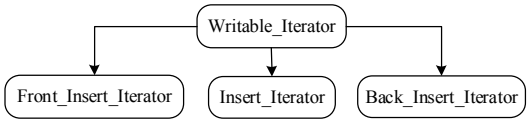

Рисунок 5 – Развитие концепций итераторов вставки

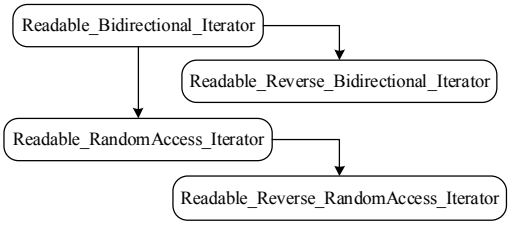

Рисунок 6 – Развитие концепций реверсивных неизменяющих итераторов

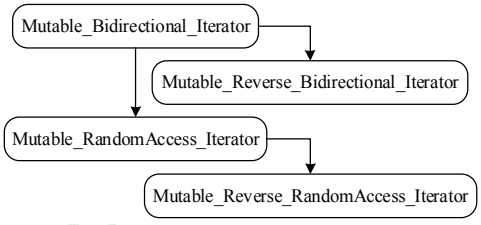

Рисунок 7 – Развитие концепций реверсивных изменяющих итераторов

Обобщённые концепции адаптеров контейнерных классов (рисунки 8 – 9) – это механизм адаптации коллекций, т.е. хранилищ, поддерживающих различные способы накопления и упорядочения объектов. Цель таких адаптеров – формализация в терминах базовых абстракций наиболее часто употребляемых операций добавления нового элемента, удаления элемента или доступа к элементу с целью обеспечения наиболее эффективной работы с уже имеющимися структурами хранения [5]. Организация способа хранения данных для их последующей обработки является весьма важным этапом разработки компьютерных программ. Для реализации большинства приложений выбор соответствующей абстрактной структуры хранения и доступа к данным является единственно важным решением, так как для одних и тех же обобщённых алгоритмов различные структуры данных и средства доступа к ним обладают неодинаковой эффективностью. Можно выделить два основных направления развития концепций адаптеров контейнерных классов. Это – диапазон (Range) и последовательность (Sequence).

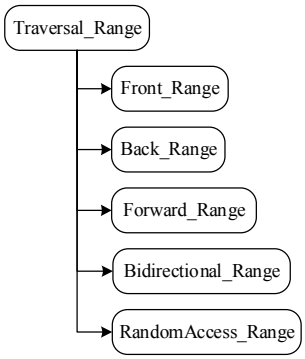

Рисунок 8 – Развитие концепций диапазонов

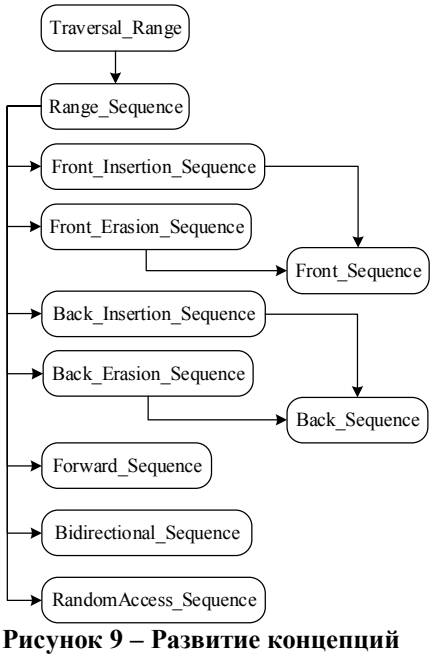

Рисунок 9 – Развитие концепций последовательностей

Функциональность, определяемая понятием «диапазон», ориентирована на доступ к элементу без возможности модификации адаптируемого набора данных (операции добавления нового элемента или удаления имеющегося). Последовательность, наоборот, позволяет модифицировать адаптируемый набор данных, но не предоставляет доступа к элементу. Такое разделение является важной составляющей при проектировании базовых абстракций как адаптеров контейнерных классов.

Для удобства организации доступа к данным было введено понятие секции, т.е. абстрактной таблицы, содержащей какой-либо набор полей и являющейся основным логическим компонентом системы при работе с данными. Эту абстракцию можно описать как группу данных, объединенных по общим признакам (полям или атрибутам) либо каким-то иным критериям общности (рисунок 10). Каждая из секций имеет свой формат обращения к данным, что определяет способ представления информации компонентами библиотеки GLEX.

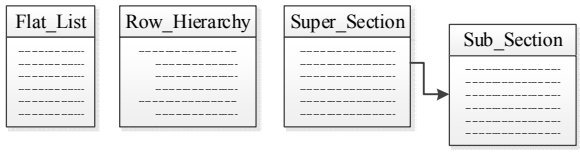

Рисунок 10 – Структурные виды секций

Можно выделить три структурных вида секций.

Плоский список (Flat_List) – набор значений атрибутов конкретной сущности.

Иерархия строк (Row_Hierarchy) – секция, строки которой ссылаются друг на друга, образуя при этом древовидную иерархическую структуру. Каждая строка может ссылаться на одну или несколько дочерних строк, что даёт возможность описывать иерархии любой сложности и любого уровня вложенности.

Подчинённая секция (Sub_Section), т.е. секция, строки которой являются дочерними по отношению к строке или строкам другой родительской секции (Super_Section). Организованные подобным образом данные могут располагаться в таблицах любых структурных видов. Иными словами, и подчинённая, и родительская секции могут иметь любой структурный вид. Они могут быть как плоским списком, так и иерархией строк.

Экспериментальные исследования

Далее приводятся фрагменты кода моделей соответствующих обобщённых концепций итераторов и их адаптеров, а также концепции разновидностей доступа к данным (Readable, Writable) для получения или установки значения элемента (рисунки 11 – 22):

```
public interface
ITraversalIterator<I extends
ITraversalIterator<I>> extends Cloneable
{
        void advance();
        I getIterator();
        ITraversalIterator<I> clone();
};
```

Рисунок 11 – Фрагмент кода Traversal_Iterator

```
public interface
IForwardIterator<I extends
IForwardIterator<I>> extends ITraversalIterator<I>
{
        boolean equalTo(I other);
        IForwardIterator<I> clone();
};
```

Рисунок 12 – Фрагмент кода Forward_Iterator

```
public interface IReadable<E>
{
        E read();
};
```

Рисунок 13 – Фрагмент кода Readable

```
public interface
IReadableIterator<I extends
IReadableIterator<I, E>, E> extends
IForwardIterator<I>, IReadable<E>
{
        IReadableIterator<I, E> clone();
};
```

Рисунок 14 – Фрагмент кода Readable_Iterator

```
public interface IWritable<E>
{
        void write(E element);
};
```

Рисунок 15 – Фрагмент кода Writable

```
public interface
IWritableIterator<I extends
IWritableIterator <I, E>, E> extends
ITraversalIterator<I>, IWritable<E>
{
        IWritableIterator <I, E> clone();
};
```

Рисунок 16 – Фрагмент кода Writable_Iterator

```
public interface
IMutableForwardIterator<I extends
IMutableForwardIterator<I, E>, E> extends
IForwardIterator<I>, IReadable<E>, IWritable<E>
{
        IMutableForwardIterator<I, E> clone();
};
```

Рисунок 17 – Фрагмент кода Mutable_Forward_Iterator

```
public interface
IBidirectionalIterator<I extends
IBidirectionalIterator<I>> extends
IForwardIterator<I>
{
        void retreat();
        IBidirectionalIterator<I> clone();
};
```

Рисунок 18 – Фрагмент кода Bidirectional_Iterator

```
public interface
IReadableBidirectionalIterator<I extends
IReadableBidirectionalIterator<I, E>, E> extends
IBidirectionalIterator<I>, IReadable<E>
{
        IReadableBidirectionalIterator<I, E> clone();
};
```

Рисунок 19 – Фрагмент кода Readable_Bidirectional_Iterator

```
public interface
IRandomAccessIterator<I extends
IRandomAccessIterator<I, D>, D extends
Number> extends IBidirectionalIterator<I>
{
        D position();
        D sub(I other);
        void advance(D distance);
        void retreat(D distance);
        boolean lessThan(I other);
        IRandomAccessIterator<I, D> clone();
};
```

Рисунок 20 – Фрагмент кода RandomAccess_Iterator

```
public interface
IMutableRandomAccessIterator<I extends
IMutableRandomAccessIterator<I, D, E>, D extends
Number, E> extends IRandomAccessIterator<I, D>,
IReadable<E>, IWritable<E>
{
        E get(D position);
        void put(D distance, E element);
        IMutableRandomAccessIterator<I, D, E> clone();
};
```

Рисунок 21 – Фрагмент кода Mutable_RandomAccess_Iterator

```
public interface
IMutableReverseRandomAccessIterator<I extends
IMutableReverseRandomAccessIterator<I, D, E>,
D extends Number, E> extends
IMutableRandomAccessIterator<I, D, E>
{
        IMutableReverseRandomAccessIterator<I, D, E>
        clone();
        IMutableRandomAccessIterator<I, D, E> base();
};
```

Рисунок 22 – Фрагмент кода Mutable_Reverse_RandomAccess_Iterator

Ниже приведены примеры кода моделей концепций диапазонов и последовательностей как адаптеров контейнерных классов (рисунки 23 – 26):

```
public interface
ITraversalRange<R extends
ITraversalRange<R, S>, S extends Number> extends
Cloneable
{
        boolean equalTo(R other);
        boolean lessThan(R other);
        boolean empty();
        R getRange();
        S size();
};
```

Рисунок 23 – Фрагмент кода Traversal_Range

```
public interface
IFrontRange<R extends
IFrontRange<R, S, E>, S extends Number, E> extends
ITraversalRange<R, S>
{
        E front();
        IFrontRange<R, S, E> clone();
};
```

Рисунок 24 – Фрагмент кода Front_Range

```
public interface
ISequence<C extends
ISequence<C, S>, S extends Number> extends
ITraversalRange<C, S>
{
        S maxSize();
        S capacity();
        void clear();
};
```

Рисунок 25 – Фрагмент кода Sequence

```
public interface
IBackInsertionSequence<C extends
IBackInsertionSequence<C, S, E>, S extends
Number, E> extends IRangeSequence<C, S>
{
        void pushBack(E element);
        IBackInsertionSequence<C, S, E> clone();
};
```

Рисунок 26 – Фрагмент кода Back_Insertion_Sequence

Выводы

На результат процесса программирования всегда влияли ограничения, связанные либо с возможностями компьютеров, либо с возможностями человека. Несколько десятилетий назад главным ограничивающим фактором была низкая производительная способность компьютера. В настоящее время физические и аппаратные ограничения отошли на второй план. С более глубоким проникновением компьютеров в сферы научной деятельности, программные системы становятся более простыми для использования специалистами, но сложными по внутренней архитектуре. Считается, что наиболее действенным способом борьбы со сложностью программных продуктов является попытка построения соответствующего уровня абстракции, основанного на анализе общности и изменчивости объектов предметной области [5, 7]. Например, возможность использования иерархии полиморфных типов данных, связанных отношением наследования, а также наборов абстрактных требований к типам, моделирующих интерфейс и семантическое поведение, даст существенное преимущество при конструировании компонентов и подсистем работы с различными наборами данных, таких как коллекция строк, иерархия строк и т.п. (рисунок 27).

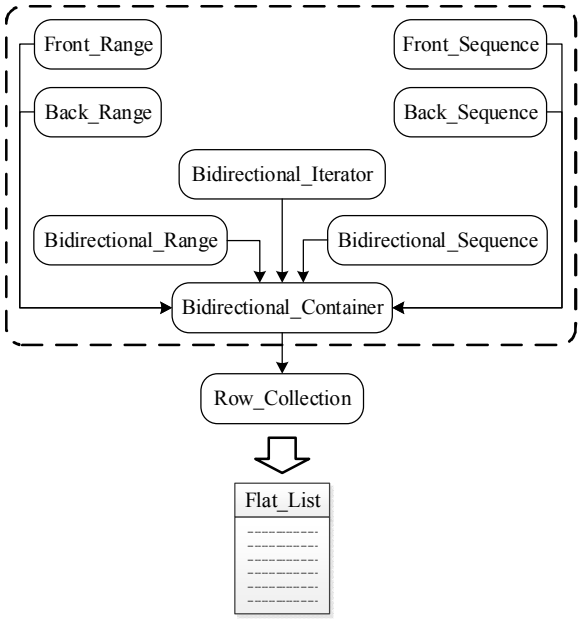

Рисунок 27 – Коллекция строк как развитие концепций диапазонов и последовательностей

Такой подход позволяет стандартизировать процессы погрузки извлечения и обработки данных в универсальном хранилище (рисунок 28).

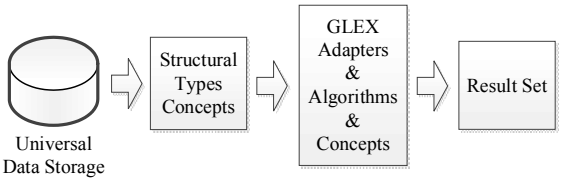

Рисунок 28 – Работа с данными, находящимися в универсальном хранилище

Предварительная реализация части компонентов библиотеки GLEX показала, что выбранный подход, предполагающий использование обобщённых концепций в качестве базовых абстракций для компонентов доступа и обработки данных, является наиболее эффективным с точки зрения снижения уровня сложности и уменьшения количества кода при автоматизации решений задач, часто встречающихся в процессе научного исследования.

Работа выполнена при поддержке Министерства образования и науки Российской Федерации, Мегагрант №14.B25.31.0005.

Библиографический список

1. **Devon M. Simmonds.** The Programming Paradigm Evolution. // IEEE Computer. 2012. no. 6. pp. 93–95.

2. **Яблоков Д. Е.** Парадигмы программирования. XIV МНПК «Научное обозрение физико-технических наук в XXI веке». Москва: Prospero, 2015. № 2 (14). С. 94-98.

3. **Яблоков Д. Е.** Использование обобщённых концепций в объектно-ориентированных языках программирования. МНТК «Перспективные информационные технологии»: Сб. науч. тр. / под ред. С. А. Прохорова. Самара: СГАУ, 2015. С. 341-345.

4. **Яблоков Д. Е.** Применение парадигмы обобщённого программирования в объектно-ориентированных языках. Информатика, моделирование, автоматизация проектирования. Сб. рекомендованных науч. тр. / под ред. Н. Н. Войта. Ульяновск: УлГТУ, 2013. С. 113-118.

5. **Блинов И. Н., Романчик В. С.** Java. Промышленное программирование. Минск: Универсал-Пресс, 2007. 704 с.

6. **Blatov V. A., Shevchenko A. P., Proserpio D. M.** Applied Topological Analysis of Crystal Structures with the Program Package ToposPro. *// Cryst. Growth Des.* 2014. no. 14 (7). pp. 3576–3586.

7. **Страуструп Б.** Язык программирования C++. Специальное издание. М.: Издательство Бином, 2011. 1136 с.

8. **Вандевурд Д., Джосаттис Н. М.** Шаблоны C++: справочник разработчика. М.: Вильямс, 2003. 544 с.

UDC 004.422+004.428

APPLYING THE GENERIC CONCEPTS AS BASE ABSTRACTIONS FOR ACCESS TO THE UNIVERSAL MODEL OF DATA STORAGE

D. E. Yablokov, Senior Developer of Samara Center for Theoretical Materials Science, PhD student of Samara University; dyablokov@gmail.com

The article deals with issues related to the creation of components of access to the universal data storage. As basic abstractions the application of generic concepts based on the use of multiple programming paradigms is proposed. The research is conducted within the project of creation an expert system using the information base of ToposPro software package. The aim of the article is to check the possibility of building the strategy of creation of independent component for access and processing the data placed in universal storage. Applying this strategy allows to connect storage structures and data accessors with the considered in the article generic concepts of adaptors of container classes and iterators. In this case the main tools dur-

ing design and implementation are generic concepts, and concrete classes based on them can be customized using overloading, aggregation, inheritance and requirements specifications to abstract data types.

Key words: *universal data model, programming paradigms, refinement of concepts, data abstraction, generic iterator concepts and container classes adapters.*

DOI: 10.21667/1995-4565-2016-57-3-68-74

Reference

1. **Devon M. Simmonds.** The Programming Paradigm Evolution. *IEEE Computer*. 2012, no. 6, pp. 93–95.

2. **Yablokov D. E.** Paradigmy programmirovaniya. *XIV MNPK «Nauchnoe obozrenie fiziko-tekhnicheskikh nauk v XXI veke»*, Moskva, Prospero. 2015, V.14, no. 2, pp. 94-98. *(in Russian).*

3. **Yablokov D. E.** Ispol'zovanie obobshchennykh kontseptsiy v ob"ektno-orientirovannykh yazykakh programmirovaniya. *MNTK «Perspektivnye informatsionnye tekhnologii». Sbornik nauchnykh trudov / pod red. S. A. Prokhorova.* Samara, SSAU, 2015, pp. 341-345. *(in Russian).*

4. **Yablokov D. E.** Primenenie paradigmy obobshchennogo programmirovaniya v ob"ektno-orientirovannykh yazykakh. *Informatika, modelirovanie, avtomatizatsiya proektirovaniya. Sbornik rekomendovannykh nauchnykh trudov / pod red. N. N. Voyta.* Ul'yanovsk, ULGTU, 2013, pp. 113-118. *(in Russian).*

5. **Blinov I. N., Romanchik V. S.** *Java. Promyshlennoe programmirovanie* (Java. Industrial programming), Minsk, UniversalPress, 2007, 704 p. *(in Russian).*

6. **Blatov V. A., Shevchenko A. P., Proserpio D. M.** Applied Topological Analysis of Crystal Structures with the Program Package ToposPro. *Cryst. Growth Des.,* 2014, no. 14 (7), pp. 3576–3586.

7. **Straustrup B.** *Yazyk programmirovaniya C++. Spetsial'noe izdanie (*Straustrup B. The C++ Programming Language. Special edition), Moscow, Izdatel'stvo Binom, 2011, 1136 p.

8. **Vandevurd D., Dzhosattis N. M.** *Shablony C++: spravochnik razrabotchika* (Vandevoort D., Josuttis N. M. Templates in C++: developer's reference), Moscow, Vil'yams, 2003, 544 p.